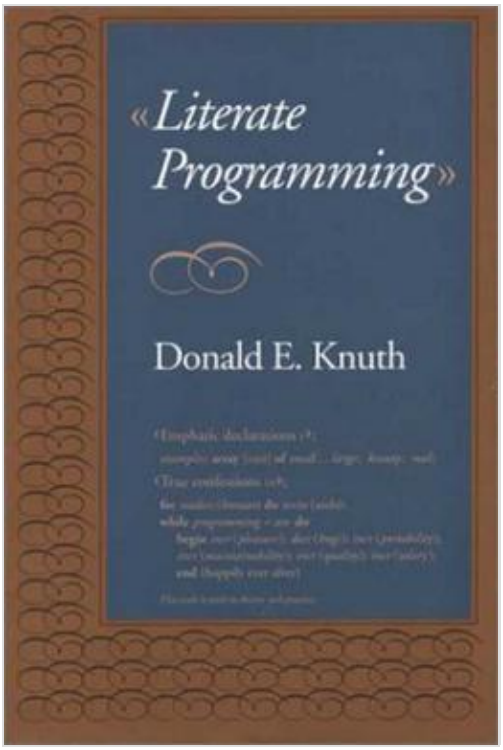
WIKIPEDIA 25
25 years of the free encyclopedia

Literate programming

Literate programming (**LP**) is a programming paradigm introduced in 1984 by Donald Knuth in which a computer program is given as an explanation of how it works in a natural language, such as English, interspersed (embedded) with snippets of macros and traditional source code, from which compilable source code can be generated.[1] The approach is used in scientific computing and in data science routinely for reproducible research and open access purposes.[2] Literate programming tools are used by millions of programmers today.[3]



Literate Programming by Donald Knuth is the seminal book on literate programming.

The literate programming paradigm, as conceived by Donald Knuth, represents a move away from writing computer programs in the manner and order imposed by the compiler, and instead gives programmers macros to develop programs in the order demanded by the logic and flow of their thoughts.[4] Literate programs are written as an exposition of logic in more natural language in which macros are used to hide abstractions and traditional source code, more like the text of an essay.

Literate programming tools are used to obtain two representations from a source file: one understandable by a compiler or interpreter, the "tangled" code, and another for viewing as formatted documentation, which is said to be "woven" from the literate source.[5] While the first generation of literate programming tools were computer language-specific, the later ones are language-agnostic and exist beyond the individual programming languages.

History and philosophy

Literate programming was first introduced in 1984 by Donald Knuth, who intended it to create programs that were suitable literature for human beings. He implemented it at Stanford University as a part of his research on algorithms and digital typography. The implementation was called "WEB" since he believed that it was one of the few three-letter words of English that had not yet been applied to computing.[6] However, it resembles the complicated nature of software delicately pieced together from simple materials.[1] The practice of literate programming has seen an important resurgence in the 2010s with the use of computational notebooks, especially in data science.

Concept

Literate programming is writing out the program logic in a human language with included (separated by a primitive markup) code snippets and macros. Macros in a literate source file are simply title-like or explanatory phrases in a human language that describe human abstractions created while solving the programming problem, and hiding chunks of code or lower-level macros. These macros are similar to the algorithms in pseudocode typically used in teaching computer science. These arbitrary explanatory phrases become precise new operators, created on the fly by the programmer, forming a *meta-language* on top of the underlying programming language.

A preprocessor is used to substitute arbitrary hierarchies, or rather "interconnected 'webs' of macros",[7] to produce the compilable source code with one command ("tangle"), and documentation with another ("weave"). The preprocessor also provides an ability to write out the content of the macros and to add to already created macros in any place in the text of the literate program source file, thereby disposing of the need to keep in mind the restrictions imposed by traditional programming languages or to interrupt the flow of thought.

Advantages

According to Knuth,[8][9] literate programming provides higher-quality programs, since it forces programmers to explicitly state the thoughts behind the program, making poorly thought-out design decisions more obvious. Knuth also claims that literate programming provides a first-rate documentation system, which is not an add-on, but is grown naturally in the process of exposition of one's thoughts during a program's creation.[10] The resulting documentation allows the author to restart their own thought processes at any later time, and allows other programmers to understand the construction of the program more easily. This differs from traditional documentation, in which a programmer is presented with source code that follows a compiler-imposed order, and must decipher the thought process behind the program from the code and its associated comments. The meta-language capabilities of literate programming are also claimed to facilitate thinking, giving a higher "bird's eye view" of the code and increasing the number of concepts the mind can successfully retain and process. Applicability of the concept to programming on a large scale, that of commercial-grade programs, is proven by an edition of TeX code as a literate program.[8]

Knuth also claims that literate programming can lead to easy porting of software to multiple environments, and even cites the implementation of TeX as an example.[11]

Contrast with documentation generation

Literate programming is very often misunderstood[12] to refer only to formatted documentation produced from a common file with both source code and comments – which is properly called documentation generation – or to voluminous commentaries included with code. This is the converse of literate programming: well-documented code or documentation extracted from code follows the structure of the code, with documentation embedded in the code; while in literate programming, code is embedded in documentation, with the code following the structure of the documentation.

This misconception has led to claims that comment-extraction tools, such as the Perl Plain Old Documentation or Java Javadoc systems, are "literate programming tools". However, because these tools do not implement the "web of abstract concepts" hiding behind the system of natural-language macros, or provide an ability to change the order of the source code from a machine-imposed sequence to one convenient to the human mind, they cannot properly be called literate programming tools in the sense intended by Knuth.[12][13]

Workflow

Implementing literate programming consists of two steps:

1. Weaving: Generating a comprehensive document about the program and its maintenance.
2. Tangling: Generating machine executable code

Weaving and tangling are done on the same source so that they are consistent with each other.

Example

A classic example of literate programming is the literate implementation of the standard Unix `wc` word counting program. Knuth presented a CWEB version of this example in Chapter 12 of his *Literate Programming* book. The same example was later rewritten for the noweb literate programming tool.[14] This example provides a good illustration of the basic elements of literate programming.

Creation of macros

The following snippet of the `wc` literate program[14] shows how arbitrary descriptive phrases in a natural language are used in a literate program to create macros, which act as new "operators" in the literate programming language, and hide chunks of code or other macros. The mark-up notation consists of double angle brackets (`<<...>>`) that indicate macros. The `@` symbol which, in a noweb file, indicates the beginning of a documentation chunk. The `<<*>>` symbol stands for the "root", topmost node the literate programming tool will start expanding the web of macros from. Actually, writing out the expanded source code can be done from any section or subsection (i.e. a piece of code designated as `<<name of the chunk>>=`, with the equal sign), so one literate program file can contain several files with machine source code.

```
The purpose of wc is to count lines, words, and/or characters in a list of files. The
number of lines in a file is ...../more explanations/

Here, then, is an overview of the file wc.c that is defined by the noweb program wc.nw:
    <<*>>=
    <<Header files to include>>
    <<Definitions>>
    <<Global variables>>
    <<Functions>>
    <<The main program>>
    @

We must include the standard I/O definitions, since we want to send formatted output
```

```
to stdout and stderr.
    <<Header files to include>>=
    #include <stdio.h>
    @
```

The unraveling of the chunks can be done in any place in the literate program text file, not necessarily in the order they are sequenced in the enclosing chunk, but as is demanded by the logic reflected in the explanatory text that envelops the whole program.

Program as a web

Macros are not the same as "section names" in standard documentation. Literate programming macros hide the real code behind themselves, and be used inside any low-level machine language operators, often inside logical operators such as `if`, `while` or `case`. This can be seen in the following `wc` literate program.[14]

```
The present chunk, which does the counting, was actually one of
the simplest to write. We look at each character and change state if it begins or ends
a word.

    <<Scan file>>=
    while (1) {
      <<Fill buffer if it is empty; break at end of file>>
      c = *ptr++;
      if (c > ' ' && c < 0177) {
        /* visible ASCII codes */
        if (!in_word) {
          word_count++;
          in_word = 1;
        }
        continue;
      }
      if (c == '\n') line_count++;
      else if (c != ' ' && c != '\t') continue;
      in_word = 0;
        /* c is newline, space, or tab */
    }
    @
```

The macros stand for any chunk of code or other macros, and are more general than top-down or bottom-up "chunking", or than subsectioning. Donald Knuth said that when he realized this, he began to think of a program as a *web* of various parts.[1]

Order of human logic, not that of the compiler

In a noweb literate program besides the free order of their exposition, the chunks behind macros, once introduced with `<<...>>=`, can be grown later in any place in the file by simply writing `<<name of the chunk>>=` and adding more content to it, as the following snippet illustrates (+ is added by the document formatter for readability, and is not in the code).[14]

```
The grand totals must be initialized to zero at the beginning of the program.
If we made these variables local to main, we would have to do this  initialization
explicitly; however, C globals are automatically zeroed. (Or rather,``statically
zeroed.'' (Get it?)

    <<Global variables>>+=
```

```
    long tot_word_count, tot_line_count,
         tot_char_count;
      /* total number of words, lines, chars */
    @
```

Record of the train of thought

The documentation for a literate program is produced as part of writing the program. Instead of comments provided as side notes to source code a literate program contains the explanation of concepts on each level, with lower level concepts deferred to their appropriate place, which allows better communication of thought. The snippets of the literate `wc` above show how an explanation of the program and its source code are interwoven. Such exposition of ideas creates the flow of thought that is like a literary work. Knuth wrote a "novel" which explains the code of the interactive fiction game Colossal Cave Adventure.[15]

Remarkable examples

- TeX and METAFONT, Knuth's respective typesetting and font description languages. Written in WEB, these were literate programming's proofs of concept.
- Physically Based Rendering[16] "describes both the mathematical theory behind a modern photorealistic rendering system and its practical implementation." This book won an Academy Award.
- Understanding MP3,[17] a complete implementation of MPEG bit streams that is also an excellent tutorial. (See also Ruckert's literate programs related to HINT.[18])
- Principia Softwarica,[19] a literate version of Plan 9 from Bell Labs.
- The Stanford GraphBase,[20] 30 short CWEB essays from Knuth that define a combinatorial computing platform, describe state-of-the-art algorithms and data structures, and provide examples of their use.
- MMIXware,[21] Knuth's software to support MMIX programming and run simulations on different architectures.
- C Interfaces and Implementations,[22] a demonstration of the titular design methodology in 24 examples.
- A Retargetable C Compiler,[23] describes the little C compiler LCC.
- Data Structures and Algorithms in C++: Pocket Primer,[24] a concise pedagogic tour.
- Inform, a language and system for writing interactive fiction. One of the largest literate programs to date, Inform is written in inweb[25] (itself a remarkable example).
- Axiom, which is evolved from scratchpad, a computer algebra system developed by IBM. It is now being developed by Tim Daly, one of the developers of scratchpad, Axiom is totally written as a literate program.[26][27]

Literate programming practices

The first published literate programming environment was WEB, introduced by Knuth in 1981 for his TeX typesetting system; it uses Pascal as its underlying programming language and TeX for typesetting of the documentation. The complete commented TeX source code was published in Knuth's *TeX: The program*, volume B of his 5-volume *Computers and Typesetting*. Knuth had

privately used a literate programming system called DOC as early as 1979. He was inspired by the ideas of Pierre-Arnoul de Marneffe.[28] The free CWEB, written by Knuth and Silvio Levy, is WEB adapted for C and C++, runs on most operating systems, and can produce TeX and PDF documentation.

There are various other implementations of the literate programming concept as given below. Many of the newer among these do not have macros and hence do not comply with the order of human logic principle, which makes them perhaps "semi-literate" tools. These, however, allow cellular execution of code which makes them more along the lines of exploratory programming tools.

Name	Supported languages	Written in	Markup language	Macros & custom order	Cellular execution	Comments
WEB	Pascal	Pascal	TeX	Yes	No	The first published literate programming environment.
CWEB	C++ and C	C	TeX	Yes	No	Is WEB adapted for C and C++.
NoWEB	Any	C, AWK, and Icon	LaTeX, TeX, HTML and troff	Yes	No	It is well known for its simplicity and for allowing text formatting in HTML rather than going through the TeX system.
Emacs org-mode	Any	Emacs Lisp	Plain text			Requires Babel,[29] which allows embedding blocks of source code from multiple programming languages[30] within one text document. Blocks of code can share data with each other, display images inline, or be parsed into pure source code using the noweb reference syntax.[31]
CoffeeScript	CoffeeScript	CoffeeScript, JavaScript	Markdown			CoffeeScript supports a "literate" mode, which enables programs to be compiled from a source document written in Markdown with indented blocks of code.[32]
Maple worksheets	Maple (software)		XML			Maple worksheets are a platform-agnostic literate programming environment that combines text and graphics with live code for symbolic computation."Maple Worksheets" (https://www.maplesoft.com/support/help/Maple/view.aspx?path=Worksheet). *MapleSoft.com*.

Name	Supported languages	Written in	Markup language	Macros & custom order	Cellular execution	Comments
						Retrieved May 30, 2020.
Wolfram Notebooks	Wolfram Language		Wolfram Language			Wolfram notebooks are a platform-agnostic literate programming method that combines text and graphics with live code.[33][34]
Jupyter Notebook, formerly IPython Notebook	Python and any with a Jupyter Kernel		JSON format Specification for ipynb (https://ipython.org/ipython-doc/3/notebook/nbformat.html)	No	Yes	Works in the format of notebooks, which combine headings, text (including LaTeX), plots, etc. with the written code.
nbdev (https://nbdev.fast.ai/)	Python and Jupyter Notebook		nbdev is a library that allows developing a python library in Jupyter Notebooks, putting all code, tests and documentation in one place.			
Julia				Pluto.jl (https://plutojl.org) is a reactive notebook environment allowing custom order. But web-like macros aren't supported.	Yes	Supports the iJulia mode of development which was inspired by iPython.
Agda						Supports a limited form of literate programming out of the box.[35]
Sweave	R		PDF			[36][37]
Knitr	R		LaTeX, PDF, LyX, HTML, Markdown, AsciiDoc, and reStructuredText			[38][39]

Name	Supported languages	Written in	Markup language	Macros & custom order	Cellular execution	Comments
Literate (https://github.com/zyedidia/Literate)	Any	D	Markdown	Yes	No	Supports TeX equations. Compatible with Vim.

Other useful tools include:

- The Leo text editor is an *outlining* editor which supports optional noweb and CWEB markup. The author of Leo mixes two different approaches: first, Leo is an outlining editor, which helps with management of large texts; second, Leo incorporates some of the ideas of literate programming, which in its pure form (i.e., the way it is used by Knuth Web tool or tools like "noweb") is possible only with some degree of inventiveness and the use of the editor in a way not exactly envisioned by its author (in modified @root nodes). However, this and other extensions (@file nodes) make outline programming and text management successful and easy and in some ways similar to literate programming.[40]
- The Haskell language has native support for semi-literate programming. The compiler–interpreter supports two file name extensions: `.hs` and `.lhs`; the latter stands for literate Haskell. The literate scripts can be full LaTeX source text, at the same time it can be compiled, with no changes, because the interpreter only compiles the text in a code environment, for example:

```
% here text describing the function:
\begin{code}
fact 0 = 1
fact (n+1) = (n+1) * fact n
\end{code}
here more text
```

The code can be also marked in the Richard Bird style, starting each line with a greater than symbol and a space, preceding and ending the piece of code with blank lines.

The LaTeX `listings` package provides a `lstlisting` environment which can be used to embellish the source code. It can be used to define a `code` environment to use within Haskell to print the symbols in the following manner:

```
\newenvironment{code}{\lstlistings[language=Haskell]}{\endlstlistings}

\begin{code}
comp :: (beta -> gamma) -> (alpha -> beta) -> (alpha -> gamma)
(g `comp` f) x = g(f x)
\end{code}
```

which can be configured to yield:

$$comp :: (\beta \to \gamma) \to (\alpha \to \beta) \to (\alpha \to \gamma)$$
$$(g \operatorname{comp} f)x = g(fx)$$

Although the package provides no means to organize chunks of code, LaTeX source code can be divided in different files.[41]

- The Web 68 Literate Programming system used ALGOL 68 as the underlying language, although there was nothing in the pre-processor 'tang' to force such use.[42]
- The customization mechanism of the Text Encoding Initiative (TEI) which enables constraining, modifying, or extending the TEI scheme, enables mixing prose documentation with fragments of schema specification in the One Document Does-it-all format. From this prose documentation, schemas, and processing model pipelines can be generated and Knuth's Literate Programming paradigm is cited as the inspiration for this way of working.[43]

See also

- Documentation generator – the inverse on literate programming where documentation is embedded in and generated from source code
- Notebook interface – virtual notebook environment used for literate programming
- Sweave and Knitr – examples of use of the "noweb"-like Literate Programming tool inside the R language for creation of dynamic statistical reports
- Self-documenting code – source code that can be easily understood without documentation
- Vibe coding

References

1. v w x y z Knuth, Donald E. (1984). "Literate Programming" (http://www.literateprogramming.com/knuthweb.pdf) (PDF). *The Computer Journal*. **27** (2). British Computer Society: 97–111. doi:10.1093/comjnl/27.2.97 (https://doi.org/10.1093%2Fcomjnl%2F27.2.97). Retrieved January 4, 2009.
2. Schulte, Eric (2012). "A Multi-Language Computing Environment for Literate Programming and Reproducible Research" (https://www.jstatsoft.org/article/view/v046i03/v46i03.pdf) (PDF). *Journal of Statistical Software*. **46** (3). doi:10.18637/jss.v046.i03 (https://doi.org/10.18637%2Fjss.v046.i03). Archived (https://web.archive.org/web/20141109113411/https://www.jstatsoft.org/article/view/v046i03/v46i03.pdf) (PDF) from the original on November 9, 2014. Retrieved May 30, 2020.
3. Kery, Mary Beth (April 2018). "The Story in the Notebook: Exploratory Data Science using a Literate Programming Tool". *CHI '18: Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery. pp. 1–11. doi:10.1145/3173574.3173748 (https://doi.org/10.1145%2F3173574.3173748).

4.

> I had the feeling that top-down and bottom-up were opposing methodologies: one more suitable for program exposition and the other more suitable for program creation. But after gaining experience with WEB, I have come to realize that there is no need to choose once and for all between top-down and bottom-up, because a program is best thought of as a web instead of a tree. A hierarchical structure is present, but the most important thing about a program is its structural relationships. A complex piece of software consists of simple parts and simple relations between those parts; the programmer's task is to state those parts and those relationships, in whatever order is best for human comprehension not in some rigidly determined order like top-down or bottom-up.

—Donald E. Knuth, *Literate Programming*[1]

5. If one remembers that the first version of the tool was called WEB, the amusing literary reference hidden by Knuth in these names becomes obvious: "Oh, what a tangled web we weave when first we practise to deceive" —Sir Walter Scott, in Canto VI, Stanza 17 of *Marmion* (1808) an epic poem about the Battle of Flodden in 1513. – the actual citation appeared as an epigraph in a May 1986 article by Jon Bentley and Donald Knuth in one of the classical "Programming Pearls" columns in *Communications of the ACM*, vol. 29, no. 5, p. 365.
6. "Literate Programming" (http://www.literateprogramming.com/knuthweb.pdf) (PDF). *Archive.ComputerHistory.org*. Retrieved June 3, 2019.
7.

> WEB's macros are allowed to have at most one parameter. Again, I did this in the interests of simplicity, because I noticed that most applications of multiple parameters could in fact be reduced to the one-parameter case. For example, suppose that you want to define something like [example elided] .... In other words, the name of one macro can usefully be a parameter to another macro.

—Donald E. Knuth, *Literate Programming*[1]

8. Knuth, Donald E.; Binstock, Andrew (April 25, 2008). "Interview with Donald Knuth" (https://www.informit.com/articles/article.aspx?p=1193856). Retrieved January 4, 2009. "Yet to me, literate programming is certainly the most important thing that came out of the TeX project. Not only has it enabled me to write and maintain programs faster and more reliably than ever before, and been one of my greatest sources of joy since the 1980s, it has actually been indispensable at times. Some of my major programs, such as the MMIX meta-simulator, could not have been written with any other methodology that I've ever heard of. The complexity was simply too daunting for my limited brain to handle; without literate programming, the whole enterprise would have flopped miserably. ... Literate programming is what you need to rise above the ordinary level of achievement."

9.

> Another surprising thing that I learned while using WEB was that traditional programming languages had been causing me to write inferior programs, although I hadn't realized what I was doing. My original idea was that WEB would be merely a tool for documentation, but I actually found that my WEB programs were better than the programs I had been writing in other languages.
>
> —Donald E. Knuth, *Literate Programming*[1]

10.

> Thus the WEB language allows a person to express programs in a *"stream of consciousness" order*. TANGLE is able to scramble everything up into the arrangement that a PASCAL compiler demands. This feature of WEB is perhaps its greatest asset; it makes a WEB-written program much more readable than the same program written purely in PASCAL, even if the latter program is well commented. And the fact that there's no need to be hung up on the question of top-down versus bottom-up, since a programmer can now *view a large program as a web, to be explored in a* ***psychologically correct order*** *is perhaps the greatest lesson* I have learned from my recent experiences.
>
> —Donald E. Knuth, *Literate Programming*[1]

11. " "Oral History of Donald Knuth"- an Interview with Ed Feigenbaum" (https://archive.computerhistory.org/resources/text/Oral_History/Knuth_Don_1/Knuth_Don.oral_history.2007.102658053_all.pdf) (PDF). *Archive.ComputerHistory.org*. Retrieved December 7, 2018.
12. Dominus, Mark-Jason (March 20, 2000). "POD is not Literate Programming" (https://www.perl.com/pub/tchrist/litprog.html/). *Perl.com*. Archived (https://web.archive.org/web/20090102151452/http://www.perl.com/pub/a/tchrist/litprog.html) from the original on January 2, 2009.

13.

> I chose the name WEB partly because it was one of the few three-letter words of English that hadn't already been applied to computers. But as time went on, I've become extremely pleased with the name, because I think that a complex piece of software is, indeed, best regarded as a web that has been delicately pieced together from simple materials. We understand a complicated system by understanding its simple parts, and by understanding the simple relations between those parts and their immediate neighbors. If we express a program as a web of ideas, we can emphasize its structural properties in a natural and satisfying way.
>
> —Donald E. Knuth, *Literate Programming*[1]

14. Ramsey, Norman (May 13, 2008). "An Example of noweb" (https://www.cs.tufts.edu/~nr/noweb/examples/wc.html). Retrieved January 4, 2009.

15. The game, also known as *ADVENT*, was originally written by Crowther in about 700 lines of FORTRAN code; Knuth recast it into the WEB idiom. It is available at literateprogramming.com (http://www.literateprogramming.com/cweb_download.html) or on Knuth's website (https://cs.stanford.edu/~knuth/programs.html) Archived (https://web.archive.org/web/20080820091137/http://sunburn.stanford.edu/~knuth/programs.html) August 20, 2008, at the Wayback Machine.
16. "Physically Based Rendering: From Theory to Implementation" (https://pbr-book.org/). *pbr-book.org*. Retrieved July 3, 2026.
17. "Homepage of Understanding MP3" (https://hint.userweb.mwn.de/understanding_mp3/index.html). *hint.userweb.mwn.de*. Retrieved July 3, 2026.
18. "Prof. Dr. Martin Ruckert - Homepage" (https://hint.userweb.mwn.de/). *hint.userweb.mwn.de*. Retrieved July 3, 2026.
19. "Principia Softwarica" (https://principia-softwarica.org/). *principia-softwarica.org*. Retrieved July 3, 2026.
20. "Knuth: The Stanford GraphBase" (https://cs.stanford.edu/~knuth/sgb.html). *cs.stanford.edu*. Retrieved July 3, 2026.
21. "MMIX Home Page" (https://mmix.cs.hm.edu/). *mmix.cs.hm.edu*. Retrieved July 3, 2026.
22. "C Interfaces and Implementations" (https://drh.github.io/cii/). *drh.github.io*. Retrieved July 3, 2026.
23. "lcc, A Retargetable Compiler for ANSI C" (https://drh.github.io/lcc/). *drh.github.io*. Retrieved July 3, 2026.
24. "Data Structures and Algorithms : Pocket Primer" (https://www.amazon.com/Data-Structures-Algorithms-Pocket-Primer/dp/1683920848). *Amazon.com*. Retrieved July 4, 2026.
25. "Welcome to the Inweb Source" (https://ganelson.github.io/inweb/index.html).
26. Page, William S. (September 2007). "AXIOM: open source computer algebra system" (https://dl.acm.org/doi/10.1145/1358190.1358206). *ACM Communications in Computer Algebra*. **41** (3): 114. doi:10.1145/1358190.1358206 (https://doi.org/10.1145%2F1358190.1358206). ISSN 1932-2240 (https://search.worldcat.org/issn/1932-2240).
27. Jenks, Richard D.; Sutor, Robert S. (1992). *Axiom: the scientific computation system*. New York : Downer's Grove, Ill., USA: Springer-Verlag; Marketed and distributed by the Numerical Algorithms Group. ISBN 978-0-387-97855-0.
28. de Marneffe, Pierre Arnoul (December 1973). Holon Programming – A Survey (https://github.com/holon-scribe/holon-programming) (Report). Université de Liège, Service d'Informatique. p. 135 – via GitHub.
29. "Introducing Babel" (https://orgmode.org/worg/org-contrib/babel/intro.html). *Orgmode.org*. Retrieved July 4, 2026.
30. "Babel Languages: redirect" (https://orgmode.org/worg/org-contrib/babel/languages.html#langs). *OrgMode.org*.
31. "Introducing Babel" (https://orgmode.org/worg/org-contrib/babel/intro.html#literate-programming). *Orgmode.org*. Retrieved July 4, 2026.
32. Ashkenas, Jeremy. "Literate CoffeeScript" (https://coffeescript.org/#literate). Retrieved November 13, 2014.
33. Milestones in Computer Science and Information Technology (https://archive.org/details/milestonesincomp0000reil/page/157) by Edwin D. Reilly, p. 157.
34. "Wolfram Notebooks" (https://www.wolfram.com/notebooks/). *Wolfram.com*. Retrieved November 28, 2018.
35. "Literate Agda" (https://wiki.portal.chalmers.se/agda/pmwiki.php?n=Main.LiterateAgda). *Agda Wiki*. Retrieved March 26, 2017.

36. Leisch, Friedrich (2002). "Sweave, Part I: Mixing R and LaTeX: A short introduction to the Sweave file format and corresponding R functions" (https://cran.r-project.org/doc/Rnews/Rnews_2002-3.pdf) (PDF). *R News*. **2** (3): 28–31. Retrieved January 22, 2012.
37. Pineda-Krch, Mario (January 17, 2011). "The Joy of Sweave – A Beginner's Guide to Reproducible Research with Sweave" (https://www.math.ualberta.ca/~mlewis/links/the_joy_of_sweave_v1.pdf) (PDF). Retrieved January 22, 2012.
38. Xie, Yihui (2015). *Dynamic Documents with R and knitr, 2nd Edition* (http://www.crcpress.com/product/isbn/9781498716963). Chapman & Hall/CRC. ISBN 9781498716963.
39. Xie, Yihui. "knitr: A General-purpose Tool for Dynamic Report Generation in R" (https://github.com/yihui/knitr/releases/download/doc/knitr-manual.pdf) (PDF) – via GitHub.
40. Ream, Edward K. (September 2, 2008). "Leo's Home Page" (http://leoeditor.com/). Retrieved April 3, 2015.
41. See listings manual (https://mirrors.ctan.org/macros/latex/contrib/listings/listings.pdf) for an overview.
42. Mountbatten, Sian. "Web 68: Literate programming with ALGOL 68" (https://web.archive.org/web/20130120154432/http://www.poenikatu.co.uk/html/Web68man.html). Archived from the original (http://www.poenikatu.co.uk/html/Web68man.html) on January 20, 2013. Retrieved January 1, 2013.
43. "TEI Guidelines" (https://web.archive.org/web/20180822123639/http://www.tei-c.org/release/doc/tei-p5-doc/en/html/USE.html#IMGD). *TEI-C.org*. TEI Consortium. Archived from the original (http://www.tei-c.org/release/doc/tei-p5-doc/en/html/USE.html#IMGD) on August 22, 2018. Retrieved August 23, 2018.

Further reading

- Sewell, Wayne (1989). "Weaving a Program: Literate Programming in WEB" (https://archive.org/details/weavingprogramli0000sewe). Van Nostrand Reinhold. ISBN 0-442-31946-0.
- Knuth, Donald E. (1992). *Literate Programming*. Stanford University Center for the Study of Language and Information. ISBN 978-0-937073-80-3.
- Gurari, Eitan M. (1994). *TeX & LaTeX: Drawing and Literate Programming*. McGraw Hill. ISBN 0-07-911616-7. (includes software)
- Nørmark, Kurt (August 13, 1998). "Literate Programming – Issues and Problems" (https://www.cs.aau.dk/~normark/litpro/issues-and-problems.html). University of Aalborg.
- Schulte, Eric (January 2012). "A Multi-Language Computing Environment for Literate Programming and Reproducible Research" (https://www.jstatsoft.org/article/view/v046i03/v46i03.pdf) (PDF). *Journal of Statistical Software*. Vol. 46, no. 3. doi:10.18637/jss.v046.i03 (https://doi.org/10.18637%2Fjss.v046.i03).
- Mall, Daniel. "Literate Programming" (http://literateprogramming.com/).
- Walsh, Norman (October 15, 2002). *Literate Programming in XML* (https://web.archive.org/web/20080511181057/http://www.idealliance.org/papers/xml02/dx_xml02/papers/06-01-01/06-01-01.html). XML 2002. CiteSeerX 10.1.1.537.6728 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.537.6728). Archived from the original (http://www.idealliance.org/papers/xml02/dx_xml02/papers/06-01-01/06-01-01.html) on May 11, 2008.

External links

- LiterateProgramming (https://wiki.c2.com/?LiterateProgramming) at WikiWikiWeb
- Literate Programming FAQ (https://www.ctan.org/tex-archive/help/comp.programming.literate_FAQ) at CTAN

Retrieved from "https://en.wikipedia.org/w/index.php?title=Literate_programming&oldid=1362512223"